```
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Module: Initialization
;>
;>      This module contains the routines that are responsible for
;>      initializing widget.
;>
;>      RESIDENT PROCEDURE RegTest
;>      RESIDENT PROCEDURE StackTest
;>      RESIDENT FUNCTION RamTest : BOOLEAN
;>      RESIDENT FUNCTION EpromTest : BOOLEAN
;>      START PROCEDURE : Main
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

                .ORG    $0

Int_Vec0:       .DW     Vector0
Int_Vec1:       .DW     Vector1
Int_Vec2:       .DW     Vector2
Int_Vec3:       .DW     Vector3
Int_Vec4:       .DW     Vector4
Int_Vec5:       .DW     Vector5

                Di
                Srp     #Wrk_Io ;select I/O working register set
                Clr     Sph             ;initialize the stack
                Ld      Spl,#Stack_Top

                Ld      P01m,#P0_03_Adr+P0_47_Out+Stack_In+P1_Adr+Mem_Ext
                Ld      !Port0,#Not_FmenL+Not_ZTestL+Not_RdHdrH+Not_ServoRst

                Ld      P2m,#P21_In+P22_In+P26_In
                Ld      !Port2,#Not_StartL+Bsy+Z8_Mem+DrwL_Read

                Ld      P3m,#B0_7_Ser+B1_6_Io+B3_4_Idm+B2_5_Io+Totem_Pol+Par_Off

                Ld      !rE,#.HIBYTE. RamBank0 ;select ram bank 1
                Ld      !rF,#01
                Lde     @!!rE,!rE
                Inc     !rE ;set next Ram adr bit
                Ld      !rF,#00
                Lde     @!!rE,!rE

                Ld      !rE,#.HIBYTE. $1000 ;save PwrFlg regs temporarily
                Ld      !rF,#.LOWBYTE. $1000
                Ld      !rD,#4 ;move 4 registers
                Ld      !rC,#PwrFlg0
Start_Loop:     Ldei    @!!rE,@!rC
                Djnz    !rD,Start_Loop

                .LSTOFF
                .DO     External
                .LSTON
```

```
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Diagnostic: Internal Ram/Register Test
;>
;>         This test is used as both a diagnostic and a selftest routine
;>      in the Widget controller. It is intended to check the internal
;>      functions of the Z8 before allowing the controller to execute
;>      any code that could be potentially dangerous to the system.
;>
;>         The test begins by checking working register set 0, registers
;>      4,5,6, and 7, then uses those registers to test the rest of the others.
;>
;>      Test: All Zeros/ All Ones
;>
;>      k := 0    { Load Value }
;>      For I := 1 To 2 Do
;>       Begin
;>        i := 120  { 128 regs: -4 i/o ports, -4 regs used in test }
;>        j := 127  { Highest register address }
;>        While ( i > 0 ) Do
;>         Begin
;>          RamReg[ j ] := k
;>          j := j - 1
;>          i := i - 1
;>         End { While }
;>        For i := 1 To 120 Do
;>         Begin
;>          If RamReg[ j ] <> k Then HALT
;>          j := j + 1
;>         End
;>        k := $FF { do all one's test next }
;>       End { for }
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Start_RegTest:

                Clr     !r4
                Or      !r4,!r4         ;quick check
                Jr      Nz,.PC.         ;loop here if can't clear r4

                Ld      !r5,!r4         ;bootstrap a few registers
                Ld      !r6,!r4         ;    /|\
                Ld      !r7,!r4         ;     |
                Ld      !r8,!r4         ;     |
                Add     !r5,!r4         ;     | ( all zeros )
                Add     !r6,!r5         ;     |
                Add     !r7,!r6         ;     |
                Add     !r8,!r7         ;    \|/
                Jr      Nz,.PC.         ;loop here if can't bootstrap

                Ld      !r4,#$FF        ;quick check
                Cp      !r4,#$FF
                Jr      Nz,.PC.         ;loop here if can't set r4

                Ld      !r5,!r4         ;bootstrap a few registers
```

```
                La      !r0,!r4         ;    /|\
                Ld      !r7,!r4         ;     |
                Ld      !r8,!r4         ;     |
                Add     !r5,!r4         ;     | ( all ones )
                Add     !r6,!r5         ;     |
                Add     !r7,!r6         ;     |
                Add     !r8,!r7         ;    \|/
                Add     !r8,#5          ; 5*(-1) + 5 = 0
                Jr      Nz,.PC.         ;loop here if can't bootstrap

                .Page

;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Test the rest of the registers
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

                Clr     !r4             ;Load value
                Ld      !r5,#RegLpTimes

RegLp1:         Ld      !r8,!r4         ;remember load value
                Ld      !r6,#RegCount-RegUsed-1_0RegUsed
                Ld      !r7,#HiRegAdr

RegLp2:         Ld      @!r7,!r4        ;begin loading ram
                Dec     !r7             ;point to new reg
                Djnz    !r6,RegLp2

                Ld      !r6,#RegCount-RegUsed-1_0RegUsed

RegLp3:         Inc     !r7
                Ld      !r4,@!r7        ;Read regs
                Cp      !r4,!r8
                Jr      Nz,.PC.         ;Loop here if failure

                Djnz    !r6,RegLp3

                Ld      !r4,#$FF        ;set up for all ones test
                Djnz    !r5,RegLp1

                .LSTOFF
                .DO     External
                .LSTON
                .Page

;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Diagnostic: Stack Test
;>
;>      This test is used to test the Z8's push and pop capabilities
;>      as well as it's ability to perform calls and returns.
;>
;>      The stack is set to internal, and the stack top is set to
;>      location $80. The ensuing PUSH instruction first decrements the
;>      stack pointer to $7F, and then store the contents of the
;>      register being pushed in that location. A pop instruction
;>      should first load the register with the contents pointed
;>      to by the stack pointer and then increment the pointer.
;>
;>      Registers Used: r4, r5
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

                Clr     Sph
                Ld      Spl,#$80

                Ld      !r4,Sph
                Jr      Nz,.PC.         ;check the loading of Stack Pointer

                Cp      Spl,#$80
                Jr      Nz,.PC.

                Ld      !r4,#$AA         ;Push/Pop $AA
                Push    !r4
                Ld      !r4,Sph
                Jr      Nz,.PC.         ;check for decrement of pointer

                Cp      Spl,#$7F
                Jr      Nz,.PC.

                Pop     !r4
                Cp      !r4,#$AA
                Jr      Nz,.PC.

                Ld      !r4,Sph
                Jr      Nz,.PC.         ;check for increment of pointer

                Cp      Spl,#$80
                Jr      Nz,.PC.

                Call    Stk_Test        ;check call capability
Stk_Test:       Ld      !r4,Spl
Stk_Test1:      Cp      !r4,#$7E        ;check for double dec on call
                Jr      Nz,.PC.

                Inc     !r4             ;point to low address byte
                Ld      !r5,@!r4        ;load low address byte
                Cp      !r5,#StkTest    ;should be next instruction after call
                Jr      Nz,.PC.

                Ld      @!r4,#Stk_Ret   ;check return capability
                Ret
Stk_Halt:       Jr      Stk_Halt        ;Halt if pc just got incremented

Stk_Ret:        Clr     Sph             ;initialize the stack
                Ld      Spl,#Stack_Top

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Initialize the i/o port control registers
;>
;>      P01m -->
;>              Select Port 0, Bits 0:3 = Adr Bits 8:11
;>              Select Port 0, Bits 4:7 = Output Ports
;>              Select Port 1 = Adr Bits 0:7
```

```
;>              Select Extended Memory Timing
;>              Select Internal Stack
;>
;>      P2m  -->
;>              Select Port 2, Bit 2 = Input
;>              Select Port 2, Bit 6 = Input
;>              All other Port 2 Bits are outputs
;>
;>      P3m  -->
;>              Select Port 3, Bit 0 = Sio Data In
;>              Select Port 3, Bit 1 = Tin
;>              Select Port 3, Bit 2 = Input
;>              Select Port 3, Bit 3 = Input
;>              Select Port 3, Bit 4 = Dm-
;>              Select Port 3, Bit 5 = Output
;>              Select Port 3, Bit 6 = Tout
;>              Select Port 3, Bit 7 = Sio Data Out
;>              Select Sio Parity Off
;>              Select Totem-Pole outputs for Port 2
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

                Ld      !P01m_Image,#P0_03_Adr+P0_47_Out+Stack_In+P1_Adr
                Ld      !P01m_StMach,#P0_03_Out+P0_47_Out+Stack_In+P1_Tri
                Ld      P01m,!P01m_Image

                Ld      P2m,#P21_In+P22_In+P26_In

                Ld      !P3m_Image,#B0_7_Ser+B1_6_Io+B3_4_Idm+B2_5_Io+Totem_Pol+Par_Off
                Ld      !P3m_StMach,#B0_7_Ser+B1_6_Io+B3_4_Io+B2_5_Io+Totem_Pol+Par_Off
                Ld      P3m,!P3m_Image

                Ld      !Dm_Mask,#$FF-Dm
                Ld      !Start_Mask,#$FF-Not_StartL

                Ld      Ipr,#$08 ;GroupA := 0, A>B>C
                Clr     Imr     ;disallow interrupts
                Clr     Irq     ;clear any pending interrupts

                Clr     DiskStat
                Clr     Excpt_Stat ;Recovery Off
                Clr     BlkStat
                Clr     RdStat
                Clr     RdErrCnt
                Clr     WrStat
                Clr     WrErrCnt
                Clr     Cache_Index
                Ld      Seek_Type,#Access_Offset
                Ld      Data_Type,#User_Type
                Clr     SeekCount
                Clr     SeekCount+1

                Ld      !rE,#.HIBYTE. Rwi_Value ;load RWI/PC cylinder value
                Ld      !rF,#.LOWBYTE. Rwi_Value
                Ld      !rC,#Hi_Rwi_Reg
                Ldci    @!rC,@!!rE
                Ldci    @!rC,@!!rE
```

```
                Clr     Tmr ;initialize timers
                Ld      Pre1,#$3 ;Mod-64, continuos run
                Ld      T1,#143 ;interrupt every 10 ms
                Or      Tmr,#T1_CntEn + T1_Load

                Ei              ;kludge for Z8 to do polling
                Di
                Ld      Imr,#Timer1 ;allow only Timer1 interrupts

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Initial Port Assignments
;>
;>      Port 0 -->
;>              Bits 0:3 are set to Adr 8:11 via P01m
;>              Bit 4 = Servo Reset Active
;>              Bit 5 = Format Enable Inactive
;>              Bit 6 = Z8 Test Inactive
;>              Bit 7 = Read Header Inactive
;>
;>      Port 1 -->
;>              Bits 0:7 are set to Adr 0:7 via P01m
;>
;>      Port 2 -->
;>              Bit 0 = Start Inactive
;>              Bit 1 = Ecc Error Inactive
;>              Bit 2 = don't care { input }
;>              Bit 3 = Busy Active
;>              Bit 4:5 = Msel0,1: Z8 <--> Mem
;>              Bit 6 = don't care { input }
;>              Bit 7 = Disk Read/Write: Read
;>
;>      Port 3 -->
;>              Don't Care
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

                Ld      !Port0,#Not_ServoRst+Not_FmenL+Not_ZTestL+Not_RdHdrH
                Ld      !Port2,#Not_StartL+Bsy+Z8_Mem+DrwL_Read

                Srp     #Wrk_Sys      ;context switch

                Call    Clr_BankSwitch

                Or      Excpt_Stat,#PwrRst ;assume a power reset
                 Ld     !r2,#.HIBYTE. $1000 ;check saved power flags
                 Ld     !r3,#.LOWBYTE. $1000
                 Ld     !rE,#.HIBYTE. PassWord
                 Ld     !rF,#.LOWBYTE. PassWord
                 Ld     !r4,#4 ;check 4 bytes

PwrRst_Lp:       Lde    !r0,@!!r2 ;get save byte
```

```
                Ldc     !r1,@!!rE ;get a byte of password
                Incw    !!r2
                Incw    !!rE
                Cp      !r0,!r1
                Jr      Nz,Power_On
                Djnz    !r4,PwrRst_Lp

                And     Excpt_Stat,#$FF-PwrRst

Power_On:       Ld      Scr_Cntr,#.HIBYTE. 2000
                Ld      Scr_Cntr+1,#.LOWBYTE. 2000

                Ld      PwrFlg0,#$F0 ;initialize power-on flags
                Ld      PwrFlg1,#$78
                Ld      PwrFlg2,#$3C
                Ld      PwrFlg3,#$1E
                Jp      Main            ;go to main routine

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: External Ram Test
;>
;>         This test is used as both a diagnostic and a selftest routine
;>      in the Widget controller. It is intended to check the external ram
;>      of the controller before allowing the controller to execute
;>      any code that could be potentially dangerous to the system.
;>
;>      Inputs:
;>              { none }
;>
;>      Outputs:
;>              ExternalRamTest : BOOLEAN { Zero flag false if error }
;>
;>      Test: All Zeros/ All Ones
;>
;>      k := $FF    { Load Value }
;>      For j := 1 To 2 Do
;>       Begin
;>        Dec_Scr_Cntr
;>        i := RamSize
;>        RamPtr := RamSize - 1
;>        While i <> 0 Do
;>         Begin
;>          Ram[ RamPtr] := k
;>          RamPtr := RamPtr - 1
;>          i := i - 1
;>         End
;>        i := RamSize
;>        While i <> 0 Do
;>         Begin
;>          If ( Ram[ RamPtr] = k ) Then HALT
;>          RamPtr := RamPtr + 1
;>          i := i - 1
;>         End
;>        k := $00 { do all zero's test next }
;>       End
;>
;>   RamSize = 2048
;>   k = r5
```

```
;>    j = r8
;>    i = r6,r7
;>    RamPtr = rA,rB
;>    scratch reg = r4
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

RamTest:
                Ld      !r4,#$FF            ;Load value
                Ld      !r8,#RamLpTimes

RamLp1:         Call    Dec_Scr_Cnt

                Ld      !r5,!r4         ;remember load value
                Ld      !r7,#.LOWBYTE. RamSize
                Ld      !r6,#.HIBYTE. RamSize
                Ld      !rB,#.LOWBYTE. HiRamAdr ;Initialize Ram Ptr
                Ld      !rA,#.HIBYTE. ( HiRamAdr+RamOffset )

RamLp2:         Lde     @!!rA,!r4       ;begin loading ram
                Decw    !!rA            ;Point to next Ram location
                Djnz    !r7,RamLp2
                Djnz    !r6,RamLp2

                Ld      !r6,#.HIBYTE. RamSize

RamLp3:         Incw    !!rA            ;Point to next Ram location to check
                Lde     !r4,@!!rA       ;Read regs
                Cp      !r4,!r5
                Jr      Nz,RamTestExit  ;Loop here if failure
                Djnz    !r7,RamLp3
                Djnz    !r6,RamLp3

                Ld      !r4,#$00        ;set up for all ones test
                Djnz    !r8,RamLp1
RamTestExit:    Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>   Function: Eprom Test
;>
;>     This test is used as both as diagnostic for the external eprom
;>   for the Widget Controller and as a selftest routine in the same
;>   units system firmware.
;>
;>     The first location in the external eprom holds one byte that
;>   is the check byte for that eprom. the check byte is calculated by
;>   adding then shifting each byte in the eprom:
;>
;>      Inputs:
;>              TopBank : BYTE { !r0 }
;>
;>      Outputs:
```

```
;>              EpromTest : BOOLEAN { zero flag false if error }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       FOR Both Halves of the Eprom Do
;>        Sum := 0
;>        Sum := Sum + 256*Eprom[ 0 ]
;>        FOR i := 1 TO MaxEpromAddress DO
;>         Dec_Scr_Cntr
;>         Sum := Sum + Eprom[ i ]
;>        IF ( Sum = 0 )
;>         THEN EpromTest := True
;>         ELSE EpromTest := False
;>
;>      { note that the check byte is stored in the first two locations
;>        of the eprom }
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

EpromTest:
                Ld      !r4,!r0 ;make a loop counter out of highaddress

E_Lp:            Ld     !r0,!r4 ;get bank to test
                 Swap   !r0 ;turn loop count back into highaddress
                Call    LookUp_Rom ;select that bank

                Clr     !r6 ;Sum := 0
                Clr     !r7
                Ld      !r8,#.HIBYTE. EpromSize
                Ld      !r9,#.LOWBYTE. EpromSize

                Ld      !rC,#.HIBYTE. EpromOffset
                Ld      !r0,#.LOWBYTE. EpromOffset

                Ldc     !r0,@!!rC ;get Eprom[ 0 ]
                Com     !r0
                Add     !r6,!r0   ;Sum := Sum + 256*Eprom[ 0 ]

                Incw    !!rC ;point to Eprom[ 1 ]
                Dec     !r9 ; account for Eprom[ 0 ]

                Ldc     !r0,@!!rC ;get Eprom[ 1 ]
                Com     !r0
                Add     !r7,!r0   ;Sum := Sum + Eprom[ 1 ]

                Incw    !!r6 ;do two's complement arithmetic

                Incw    !!rC ;point to Eprom[ 2 ]
                Dec     !r9 ; account for Eprom[ 1 ]

Eprom_Lp:       Call    Dec_Scr_Cntr
                Ldc     !r0,@!!rC ;get Eprom[ i ]
                Incw    !!rC
                Add     !r7,!r0 ;Sum := Sum + Eprom[ i ]
                Adc     !r6,#0
                Djnz    !r9,Eprom_Lp
```

```
                Djnz    !r8,Eprom_Lp
                Jr      Nz,Eprom_End

                Djnz    !r4,E_Lp

Eprom_End:      Jp      Bank_Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: MsWait
;>
;>      This procedure is used as a software timing loop, where
;>      the busy wait length is equal to .01 * InputValue in seconds.
;>      ( one unit wait = 10 ms ).
;>
;>      Inputs:
;>              WaitLength : WORD { !!r2 }
;>
;>      Outputs: { none }
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

MsWait:
                Ld      !r0,#10 ;change LED every 100 ms
MsWait_1:       And     Irq,#$FF-Timer1 ;clear any pending times

MsWait_Lp:      Tm      Irq,#Timer1 ;wait for timer int
                Jr      Z,MsWait_Lp

                Dec     !r0
                Jr      Nz,MsWait_Dec

                Push    !r2 ;save counter
                Push    !r3
                Call    Invert_Led
                Pop     !r3
                Pop     !r2
                Ld      !r0,#10

MsWait_Dec:     Decw    !!r2 ;count down a unit
                Jr      Nz,MsWait_1

                Ret


Dec_Scr_Cntr:   Tm      Irq,#Timer1 ;wait for timer interrupt
                Jr      Z,Dec_Scr_End

                Call    Invert_Led
                And     Irq,#$FF-Timer1 ;get rid of old interrupt
                Ld      !r0,Scr_Cntr ;check for already zero count
                Or      !r0,Scr_Cntr+1
                Jr      Z,Dec_Scr_End
```

```
                Decw    Scr_Cntr
Dec_Scr_End:    Ret


Clr_BankSwitch: Ld      !r2,#.HIBYTE. BankReg
                Ld      !r3,#.LOWBYTE. BankReg
                Ld      !r1,#7 ;clear 7 bits
Clr_B_Lp:       Lde     @!!r2,!r2
                Inc     !r2 ;next bit's adr is $100 above last one
                Djnz    !r1,Clr_B_Lp

                Ld      !r0,Excpt_Stat ;clear all but LED state
                And     !r0,#$FF-Led_Mask
                Call    Set_Led

                 Ld     !r0,#Ram1
                Call    Set_RamBank
                Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Start_Procedure: Main
;>
;>      Inputs: { none }
;>
;>      Outputs: { none }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       SlfTst_Result := { assume all tests fail }
;>       SlfTst_Result.RamTest := RamTest
;>       SlfTst_Result.EpromTest := EpromTest
;>       Init_ExtStack
;>       ZeroBlock( ReadArray )
;>       ClearStatus
;>       Load_PassWord( WBlkFence )
;>       SelfTest
;>       IF system passed selftest
;>        THEN
;>             Recovery is turned ON
;>             Load_SprTbl
;>             Load_Cache
;>             IF SystemCode THEN Scan
;>        ELSE Recovery is turned OFF
;>       Clr_Bsy( Init_Response, Not( Cmnd_Pending )
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      MAIN
```

```
;?
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Main:
                Ld      SlfTst_Result,#$FF ;assume all failures at first

                Call    RamTest         ;test external ram
                Jr      Nz,Main_Eprom
                And     SlfTst_Result,#$FF-Ram_Fail

Main_Eprom:      Ld     !r0,#Eprom2
                Call    EpromTest       ;test external eprom bank 0,1
                Jr      Nz,Main_SelfTst
                And     SlfTst_Result,#$FF-Eprom_Fail

Main_SelfTst:    Call    ExtStk_Vector ;init external stack

                 Ld     !rE,#.HIBYTE. WBlkFence
                 Ld     !rF,#.LOWBYTE. WBlkFence
                Call    LdPw_Vector ;set write buffer fence

                Call    ClrStat_Vector ;clear all status'

                Call    SlfTst_Vector ;perform selftest

Main_LdSpr:     Tm      SlfTst_Result,#$FF-No_SprTbl
                Jr      Z,Main_Set_R
                And     Excpt_Stat,#$FF-Recovery ;THEN Recovery is off
                Jr      Main_Lp1

Main_Set_R:     Or      Excpt_Stat,#Recovery ;ELSE it is ON

                Call    SprTbl_Vector

Main_Cache:     Call    LC_Vector

                Call    IScan_Vector

Main_Lp1:       Call    Clr_BankSwitch
                Call    Set_SeekNeeded

                 Ld    Wrk_Io+$A,#Init_Response
                 Clr   Wrk_Io+$B ;Cmnd_Pending, IBsy := False

;               \ /     ;in-line code, must be followed by Clr_Bsy

                .LSTOFF
```